An Initial Analysis of the Impact of Overlap and Independent Progress for MPI

Ron Brightwell, Keith D. Underwood, and Rolf Riesen

Sandia National Laboratories[⋆]
PO Box 5800, MS-1110
Albuquerque, NM 87185-1110
{rbbrigh,kdunder,rolf}@sandia.gov

Abstract. The ability to offload functionality to a programmable network interface is appealing, both for increasing message passing performance and for reducing the overhead on the host processor(s). Two important features of an MPI implementation are independent progress and the ability to overlap computation with communication. In this paper, we compare the performance of several application benchmarks using an MPI implementation that takes advantage of a programmable NIC to implement MPI semantics with an implementation that does not. Unlike previous such comparisons, we compare identical network hardware using virtually the same software stack. This comparison isolates these two important features of an MPI implementation.

1 Introduction

Two desirable qualities of an MPI implementation are the ability to efficiently make progress on outstanding communication operations and the ability to overlap computation and communication. Unfortunately, it is difficult to isolate these features in order to determine how much of an effect they have on overall performance or to use them to characterize applications that may be able to take advantage of the opportunity for performance improvement that they offer. This is especially difficult when comparing application performance with different networks using identical compute hardware. For example, there have been several extensive comparisons of high-performance commodity interconnects, such as Myrinet [1], Quadrics [2], and InfiniBand [3]. These comparison typically use micro-benchmarks to measure raw performance characteristics and application benchmarks or applications to measure whole system performance. Unfortunately, this type of analysis can still leave many important questions unanswered.

Implementations of MPI typically strive to utilize the capabilities of the underlying network to the fullest and exploit as many features as possible. In this paper, we do the opposite. We purposely limit the capabilities of an MPI implementation in order to try to better understand how those specific capabilities impact application performance. Rather than simply using identical compute hardware, we use identical networking hardware and a significant portion of the same network software stack. This

[⋆] Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

approach allows us to compare two different MPI implementations that have nearly identical micro-benchmark performance only on the basis of the features that differ.

The rest of this paper is organized as follows. The following section provides additional background on the features that we have isolated in this experiment. Section 3 provides an overview of research that motivated this effort while Section 4 discusses how this work complements other previous and ongoing projects in this area. Section 5 describes the specific software and hardware environment used in this study, and Section 6 presents an analysis of the data. The conclusions of this paper are presented in Section 7, which is followed by an outline of future work.

2 Background

The MPI Standard [4] defines a Progress Rule for asynchronous communication operations. The strict interpretation of this rule states that once a non-blocking communication operation has been posted, a matching operation will allow the operation to make progress regardless of whether the application makes further library calls. For example, if rank 0 posts a non-blocking receive and performs an infinite loop (or a significantly long computation) and rank 1 performs a matching blocking send operation, this operation will complete successfully on rank 1 regardless of whether or not rank 0 makes another MPI call. In short, this rule mandates non-local progress semantics for all non-blocking communication operations once they have been enabled.

Unfortunately, there is enough ambiguity in the standard to allow for a weak interpretation of this rule. This interpretation allows a compliant implementation to require the application to make library calls in order to make progress on outstanding communication operations. Independent of compliance, it can be easily observed that an implementation that adheres to the strict interpretation offers an opportunity for a performance advantage over one that supports the weak interpretation. One of the goals of this work is to better understand the impact of independent progress, where an intelligent and/or programmable network interface is responsible for making progress on outstanding communications independent of making MPI library calls.

It is possible to support overlap without supporting independent MPI progress. Networks capable of performing RDMA read and write operations can fully overlap communication with computation. However, the target address of these operations must be known. If the transfer of the target address depends on the user making an MPI library call (after the initial operation has begun) then progress is not independent. If the transfer of the target address is handled directly by the network interface, or by a user-level thread, then independent progress can be made. Conversely, it is possible to have independent progress without overlap. An example of this is the implementation of MPI for ASCI Red [5], where the interrupt-driven nature of the network interface insures that progress is made, but the host processor is dedicated to moving data to the network (at least in the default mode of operation where the communication co-processor is not used).

3 Motivation and Approach

Previous work[6] has indicated that a number of the NAS parallel benchmarks use excessively long posted receive and unexpected message queues in the MPI library. We hypothesized that this could degrade performance on platforms that offload the traversal of these queues onto much slower network interface hardware. The Quadrics environment provides the ideal opportunity to evaluate this hypothesis as it offers a native MPI implementation that offloads the traversal of these queues onto a 100 MHz processor (in the case of the ELAN-3 hardware) processor. Quadrics also offers a Cray SHMEM [7] compatibility interface, which can be used as a lightweight layer for building an MPI library without offloading any of the traditional MPI queue management onto the NIC [**?**]. Micro-benchmarks[8] indicate that, indeed, message latency is dramatically smaller on an MPI built over SHMEM when queue lengths grow. This work seeks to determine if application performance correlates to that finding.

A second motivation for comparing MPI libraries built on SHMEM and TPorts is that it allows the comparison of a library that provides capabilities for computation and communication overlap as well as independent progress in MPI to a library that does not. The MPI implementation using Quadrics Tports provides independent progress and overlap, since a thread running on the NIC is able to respond to incoming MPI requests without host processor intervention. In contrast, the SHMEM interface still allows messages to be deposited directly into user memory without intervention by the host, but a host CPU must still be used to handle MPI queue management and protocol messages. Using the host processor removes the opportunity for significant overlap and does not allow for independent progress of outstanding communication requests. Thus, a single platform with a similar network software stack can be used to evaluate both approaches.

Overlap and independent progress are believed to be important performance enhancements, but it is seldom possible to evaluate the relative merits of each on identical hardware. By reducing the differences in the system to the MPI implementation, this work is able to evaluate these issue. This has important implications for newer parallel computer networks, such as InfiniBand [3], that use RDMA for MPI implementations. Most implementations of MPI for InfiniBand [**?**,9, 10] do not efficiently support overlap or independent progress [11].

Finally, benchmarks are notorious for their insensitivity to critical system parameters that ultimately impact system performance; however, “real” applications can require person months of effort to port to new platforms. In contrast, benchmarks are typically easy to port to new environments. Thus, the comparison of the NAS parallel benchmarks on two different types of MPI implementations on a single hardware platform will help to reveal whether they test certain system parameters of interest (in this case, the impact of queue usage and the impact of independent progress in MPI).

4 Related Work

The work in this paper spans the areas of performance analysis, NIC-based protocol offload, and characterizing the behavior of parallel applications. However, we know of no work that characterizes the benefits and drawbacks of NIC offload using identical hardware and nearly identical software stacks.

There is an abundant amount of work that compares different network technology using identical compute hardware in order to evaluate networks as a whole. The most recent and comprehensive study for MPI and commodity HPC networks is [11]. The work we present here attempts to isolate specific properties of a network - host processing versus NIC offload - in order to evaluate the two strategies.

It is typical for papers that describe MPI implementations to explore different strategies within the implementation. An example of this is [12], where polling versus blocking message completion notification is explored. This kind of MPI implementation strategy work explores different methods of using the capabilities offered by a network, but limiting network capabilities in order to characterize specific network functionality has not been explored.

A desciption of the MPI implementation for Cray SHMEM as well as communication micro-benchmark performance is described in [**?**],

5 Platform

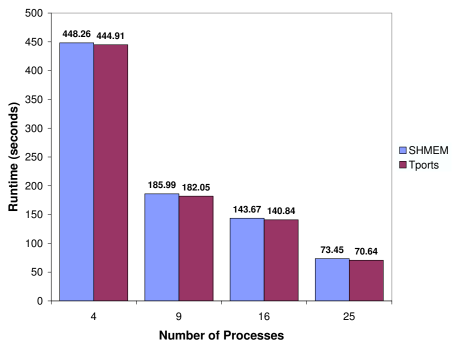
The machine used for our experiments is a 32-node cluster at Los Alamos National Laboratory. Each node in the cluster contains two 1 GHz Intel Itanium-2 processors, 2 GB of main memory, and two Quadrics QsNet (ELAN-3) network interface cards. The nodes were running a patched version of the Linux 2.4.21 kernel. We used version 1.24-27 of the QsNet MPI implementation and version 1.4.12-1 of the QsNet libraries that contained the Cray SHMEM compatibility library. The SHMEM MPI library is a port of MPICH [13] version 1.2.5. All applications were compiled using Version 7.1 Build 20031106 of the Intel compiler suite.

All of our experiments were run using only one process per node using only one network interface, and all results were gathered on a dedicated machine. In order to provide a fair comparison, we disabled use of the optimized collective operations for the Tports MPI runs so that both implementations used the collective operations layered over MPI peer communication operations provided by default with MPICH. Ultimately, this change had negligible impact on the performance of the benchmarks using the Tports MPI.

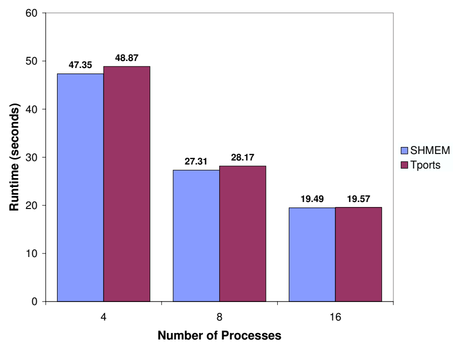
6 Results

Here we compare the performance of Tports MPI and SHMEM MPI for the class B NAS parallel benchmarks. EP (the embarassingly parallel benchmark) was excluded since it does not do any significant communications. Each benchmark was run 4 times for each number of processors per job. The average of the four runs is reported in Figure 1.

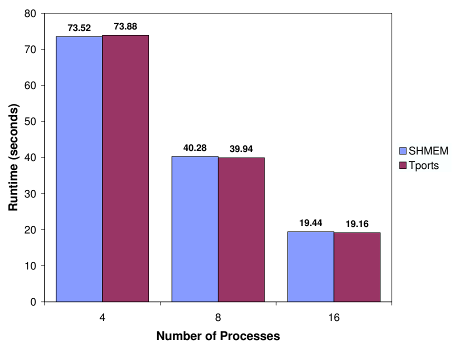
Figure 1(a) graphs the performance of BT for both MPI implementations. Tports MPI has a 2-4% advantage over SHMEM MPI, depending on the number of nodes involved. While this may seem like it could be attributed to noise in the measurements,



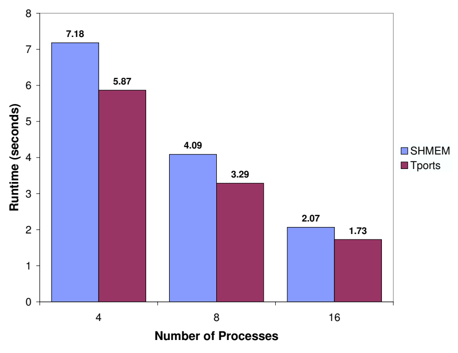
(a)



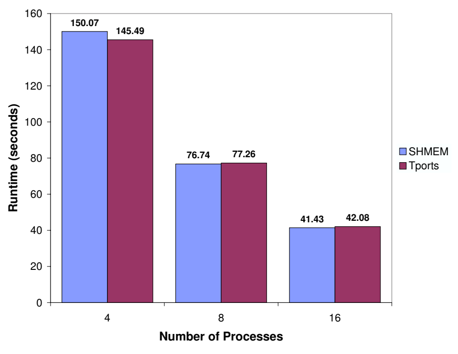
(b)



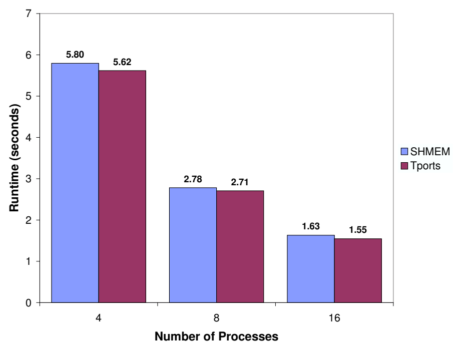
(c)



(d)



(e)



(f)

Fig. 1. A comparison of Tports and SHMEM for: **(a)** BT, **(b)** CG, **(c)** FT, **(d)** IS, **(e)** LU, and **(f)** MG

the maximum difference in the minimum and maximum run times was 0.5% for each implementation. Figure 1(b) shows the performance of CG, which is one of the few benchmarks where SHMEM MPI appears to have advantages. For small numbers of nodes, SHMEM MPI has up to a 3% performance advantage over Tports MPI with measurements that vary by less than 0.5%. This performance advantage is surprising since since CG was found to be a "well-behaved" application in terms of MPI queue usage[6]. The cause is likely to be the use of messages that are predominantly smaller than 2 KB as discussed in [11]. However, this advantage evaporates at 16 nodes.

The FT and IS benchmarks were expected to pose a significant performance issue for an MPI implementation that offloads queue handling to a slower processor on the NIC (e.g. MPI over Tports). Both of these benchmarks have long unexpected queues, long posted receive queues, and long average traversals of each[6]. Traversing these queues on an embedded processor should have proved costly; however, Figures 1(c) and (d) clearly indicate otherwise. Performance for FT is effectively equivalent between the two implementations and Tports MPI has a significant advantage (20%) for IS. Figures 1(e) and (f) continue to show roughly equivalent performance for the two MPI implementations (on the LU and MG benchmarks), but Figure 2 once again shows a significant margin of performance improvement for the Tports implementation. SP performs 5-10% better simply by using offload that provides overlap and independent progress.

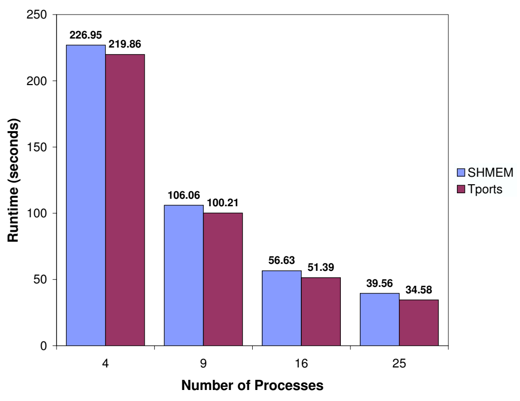


Fig. 2. A comparison of Tports and SHMEM for the SP benchmark

7 Conclusions

This study set out to evaluate the impacts of an MPI implementation that provides independent progress and overlap capabilities by using NIC based MPI offload. The expectation was that both advantages and disadvantages would be uncovered — advantages for "well-behaved" applications that had short MPI queues and disadvantages for “poorly-behaved” applications that used long MPI queues. In the process, we sought to answer three questions: does offload negatively affect applications with long MPI queues? do overlap and independent progress yield an advantage to applications? and, can the NAS benchmarks answer these questions. The answers were surprising.

The Tports MPI (the offloading MPI) wins almost uniformly (and sometimes by a significant margin) despite the fact that the SHMEM MPI has almost identical performance. In the rare cases where SHMEM MPI is faster, it is by a very small margin. In contrast, Tports MPI has a surprisingly large margin of victory over SHMEM MPI (20%) on the IS benchmark, which is “poorly-behaved”. Also surprising was the fact that SHMEM MPI had a slight edge in performance for CG (a “well-behaved” application). In summary, “poorly-behaved” applications did not suffer from the slow NIC processor that offloads MPI queue handling.

Overall, the results in this paper point to a significant advantage for the combination of independent progress and overlap in an MPI implementation. The SHMEM MPI only has an advantage that could be argued to be statistically signficant in 3 of the 23 data points compared. The largest of these margins is 3% for one data point for CG. In contrast, the Tports MPI has 5-10% advantages for all of the SP measurements, 2-4% advantages for the BT measurements, and 20% advantages for IS. Since the difference in microbenchmark performance is always smaller than this, offloading and independent progress clearly have performance benefits.

Finally, it is clear that at least 3 of 7 the NAS benchmarks see performance impacts from overlap and independent progress. This is good news for those seeking to evaluate new platforms.

8 Future Work

These results have produced two avenues of future research. First, the seeming lack of impact of queue length on NAS parallel benchmark performance for networks that perform offloading of queue traversal warrants further investigation. There are three possible causes. It is possible that message latency does not significantly affect NAS benchmark performance. It is also possible that there are other aspects of system behavior that are penalizing non-offloading implementations in a way that is not captured by microbenchmarks (e.g. the “Rogue OS” effect[14]). As a third possibility, queue behavior for the NAS benchmarks may be drastically different on Quadrics than on the Myrinet platform where we originally measured it.

The second avenue for future work will be the study of full ASCI applications. Studying full ASCI applications takes significanly more work than studying benchmarks because they can easily include 100,000 lines of code, use 2 or more programming languages, use several difficult to port third party librarys, and require non-automated adaptation of platform specific build scripts; thus, each application can take an application specialist just to build it on a new platform. Nonetheless, we believe that these preliminary results justify further study into ASCI application behavior.

9 Acknowledgments

The authors would like to gratefully acknowledge the CCS-3 group at Los Alamos National Laboratory, especially Fabrizio Petrini, for providing access to the cluster used for our experiments.

References

1. Boden, N.J., Cohen, D., Kulawik, R.E.F.A.E., Seitz, C.L., Seizovic, J.N., Su, W.K.: Myrinet: A gigabit-per-second local area network. IEEE Micro **15** (1995) 29–36
2. Petrini, F., chun Feng, W., Hoisie, A., Coll, S., Frachtenberg, E.: The Quadrics network: High-performance clustering technology. IEEE Micro **22** (2002) 46–57
3. Infiniband Trade Association: http://www.infinibandta.org. (1999)
4. Message Passing Interface Forum: MPI: A message-passing interface standard. The International Journal of Supercomputer Applications and High Performance Computing **8** (1994)
5. Brightwell, R., Shuler, P.L.: Design and implementation of MPI on Puma portals. In: Proceedings of the Second MPI Developer's Conference. (1996) 18–25
6. Brightwell, R., Underwood, K.D.: An analysis of NIC resource usage for offloading MPI. In: Proceedings of the 2004 Workshop on Communication Architecture for Clusters, Santa Fe, NM (2004)
7. Cray Research, Inc.: SHMEM Technical Note for C, SG-2516 2.3. (1994)
8. Underwood, K.D., Brightwell, R.: The impact of MPI queue usage on message latency. In: Proceedings of the International Conference on Parallel Processing (ICPP), Montreal, Canada (2004)
9. Liu, J., Chandrasekaran, B., Yu, W., Wu, J., Buntinas, D., Kini, S.P., Wyckoff, P., Panda, D.K.: Micro-benchmark performance comparison of high-speed cluster interconnects. IEEE Micro **24** (2004)
10. Rehm, W., Grabner, R., Mietke, F., Mehlan, T., Siebert, C.: An MPICH2 channel device implementation over VAPI on InfiniBand. In: Proceedings of the 2004 Workshop on Communication Architecture for Clusters. (2004)
11. Liu, J., Chandrasekaran, B., Wu, J., Jiang, W., Kini, S., Yu, W., Buntinas, D., Wyckoff, P., Panda, D.K.: Performance comparison of MPI implementations over InfiniBand, Myrinet and Quadrics. In: The International Conference for High Performance Computing and Communications (SC2003). (2003)
12. Dimitrov, R., Skjellum, A.: Impact of latency on applications' performance. In: Fourth MPI Developers' and Users' Conference. (2000)
13. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard. Parallel Computing **22** (1996) 789–828
14. Petrini, F., Kerbyson, D.J., Pakin, S.: The case of the missing supercomputer performance: Identifying and eliminating the performance variability on the ASCI Q machine. In: Proceedings of the 2003 Conference on High Performance Networking and Computing. (2003)